REMARKS

This is in response to the Office Action mailed on October 11, 2005 in which claims 28-55 were rejected. With this Amendment, claims 28 and 49 have been amended and claims 26-27 and 35 have been canceled. Claims 28-34 and 36-55 are pending in this application.

Rejections to the Specification

With this Amendment claims 26-27 that were previously withdrawn from consideration have now been canceled.

Objection to the Drawings

In the Office Action the drawings were rejected for various informalities. Five sheets of formal drawings were submitted on February 8, 2005, which overcome these informalities. The formal drawings were accepted in the Office Action mailed on April 4, 2005 (Paper No. 033105). In particular, the formal drawings do not contain a hand written label for FIG. 2, and all shading has been replaced with stippling. Therefore, the objection to the drawings should be withdrawn.

Pending Claims

The current status of the claims is that claims 1-27 have been canceled and claims 28-34 and 36-55 are pending in this application. In the Office Action, claims 54 and 55 were omitted from the Office Action Summary, and in the claim rejections under 35 U.S.C. § 102. However, the Office Action did address the grounds of rejection of claims 54 and 55 on page 9.

Claim Rejections - 35 U.S.C. § 102

In the Office Action claims 28-55 were rejected under 35 U.S.C. § 102(b) as being anticipated by the Template Software system. With this Amendment, independent claims 28 and 49 have been amended and claim 35 has been canceled.

Before specifically addressing the merits of the claims, a general description of some of the main differences between the Template Software system and the present system will be provided. One of the primary differences in the present system is that component development (compiling components that

are built to our specification) is done separately from system development. In practice, components that were developed and compiled five years ago can still be used in new systems, and no further compiling needs to take place for the new system, because system development does not involve compiling. In fact, from the system developer's perspective the only entities that exist in the system are component instances and links. Any of these can be added or deleted at any time without compiling. Template Software does not allow this functionality.

Another difference is that in the present system component instances and links exist as live objects during both design-time and run-time. During design-time, live component instances and links interact with the system development tool; during run-time they interact with the engine software program. In addition, component instance behavior is different during design-time than it is at run-time. During design-time component instances operate in an edit mode during which configuration settings are modified. During run-time component instances operate in a run mode to perform their desired functionality. In either case the component instances are living during design-time and no compiling is involved during either development or deployment. When deployed, the same component instance comes to live in the run-time environment (engine software program). This approach provides for much greater flexibility and extensibility as to the pre-defined behavior of the components.

Another difference is that system development in the present system can be performed with only three steps: (1) create component instances, (2) define links between component instances, and (3) deploy the metadata that defines component instances and links to the run-time environment. These steps do not have to be performed in a fixed sequence. That is, as soon as two component instances are created a link can be defined between them. More component instances and links can be added, deleted, or modified even after deployment. This cannot be said of the Template Software system.

Although Template Software does allow interpreted routing rules that can be changed without compiling, Template Software cannot change an entire application without compiling. In the present system any component instances and links can be changed, after the system is developed and deployed,

without compiling. Since component instances and links are the only entities that are defined when creating a system, this approach provides a higher level of flexibility. Therefore, it can be seen that significant differences exist between the present system and the system of Template Software, Inc.

With that foundation, we now turn to an examination of the claims as amended. Claims 28 and 49 have been amended to explicitly recite features that are not taught by the Template Software system. The claims identify three elements of the architecture for developing a distributed information system. The first element is a component development tool. The second element is a system development tool. The third element is an engine software program. Each of these elements perform their own unique functions within the system.

First, the component development tool generates a plurality of autonomous and compiled components that implement and consume services, the components capable of operating in an edit mode and a run mode. The two modes of operation are further discussed below. However, it is important to note that the component development tool generates a plurality of autonomous and compiled components that are later used by other elements of the system. Before being used by these other elements of the system, however, they are fully compiled and autonomous.

The Template Software system does not teach each of the features of the component development tool as recited in claims 28 and 49. For example, Template Software does not teach a component development tool for generating a plurality of autonomous and compiled components, the components capable of operating in an edit mode and a run mode. Rather, the tasks of the Template Software system operate only in a single mode. Furthermore, the Template Software system does not teach components that are autonomous and compiled before being used by the system development tool or the engine run time. This feature is identified in the claims by the fact that the component development tool generates the components, which are later instantiated and operated as defined.

Second, the system development tool defines and hosts a plurality of component instances. The component instances are capable of operating in the edit mode while hosted by the system

development tool. After the component has been generated and compiled by the system development tool, it can then be instantiated by the system development tool. The autonomous and compiled component instance then operates in the edit mode, a mode that allows a user of the system development tool to configure the component instance as desired. In addition, links between component instances are defined with the system development tool. It is important to recognize that a user of the system development tool does not need to write any code to perform these tasks.

The Template Software system does not teach each of the features of the system development tool as recited in claims 28 and 49. For example, the Template Software system does not perform all of the features of the system development tool without requiring writing of code. Rather, the Template Software system does require the writing of code which is then compiled to build an application. Evidence of this can be found for example at page 7-2 of the "Using the WFT Development Environment" document, where it defines the term "build" as "to compile the source files for a software development effort into object files that can be executed." On page 7-14 of the same document it describes how to issue the command to build (compile) the application before it is deployed.

Third, the engine software program provides a dynamic run-time environment for hosting the plurality of component instances. The component instances operate in the run mode while hosted by the engine software program. In this mode the component instances perform their desired functionality within the distributed information system. The engine software program also supports communication between component instances based upon the defined links. The defined links are defined in the system development tool without requiring the writing of any code.

The Template Software system does not teach each of the claimed elements of the engine software program as recited in claims 28 and 49. For example, the Template software system does not teach a component that operates in a run mode, distinct from an edit mode, while being hosted by the engine software program.

Thus, it can be seen that one of the benefits over the Template Software system is that once components have been generated by the component development tool, no further compiling or code writing must take place. The components are completely self-sufficient (autonomous) and operational. They are capable of operating in two different modes, depending on whether they are being hosted by the system development tool or by the engine software program.

Template Software, Inc. does not teach each and every element of independent claims 28 and 49. Therefore, these claims are in condition for allowance. Dependent claims 29-34, 36-48 and 50-55 depend from independent claims 28 and 49, and are therefore also allowable.

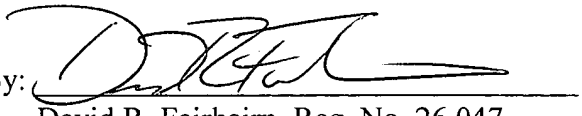
Conclusion

In view of the foregoing, this application containing pending claims 28-34 and 36-55 is in condition for allowance. Reconsideration and notice to that effect is respectfully requested.

Respectfully submitted,

KINNEY & LANGE, P.A.

Date: 2/13/06

By: 

David R. Fairbairn, Reg. No. 26,047
THE KINNEY & LANGE BUILDING
312 South Third Street
Minneapolis, MN 55415-1002
Telephone: (612) 339-1863
Fax: (612) 339-6580

DRF:BAT:amy